```
105:  Account1(){

110:        P1();

115:        Status = P2(Temp);

120:        if (Status==ERROR1){
125:              do-reverse-of-P2();
130:              do-reverse-of-P1();
135:              exit ();
138:        }

139:        temp-was-one=0;
140:        If (Temp==1){
145:              P3();
148:              P4();
150:              P5();
152:              temp-was-one=1;
153:        }


155:        Status=P6();

160:        if (status==ERROR2)  {
165:              do-reverse-of-P6();
170:              If (temp-was-one==1){
175:                    do-reverse-of-P5();
178:                    do-reverse-of-P4();
180:                    do-reverse-of-P3();
183:              }
185:              do-reverse-of-P2();
190:              do-reverse-of-P1();
195:              exit ();
197:        }
199:  }
```

FIG. 1
(Prior Art)

```
205: Account1()
206:    {
207:    Typedef int (*fptr)();
208:    Tstep(int, fptr, fptr);
209:    int txid = Tinit();


215:            Tstep (txid, P1(), R1());

225:            Status =  Tstep (txid, P2(Temp), R2());
230:            if (Status == ERROR) {
240:                    Tabort (txid);
245:                    exit();
247:            }

250:            if (Temp==1) {
255:                    Tstep (txid, P3(), R3());
260:                    Tstep (txid, P4(), R4());
265:                    Tstep (txid, P5(), R5());
267:            }

270:            Status = Tstep (txid, P6(), R6());

275:            if (Status == ERROR) {
280                     Tabort (txid);
285:            exit();
290:            }

295:    Tcommit (txid)
299:    }
```

FIG. 2

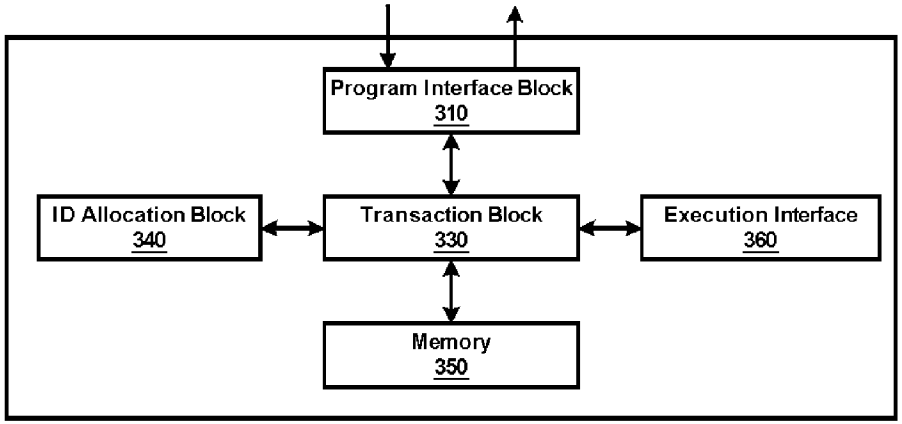
Program Interface Block
310
ID Allocation Block
340
Transaction Block
330
Execution Interface
360
Memory
350

FIG. 3

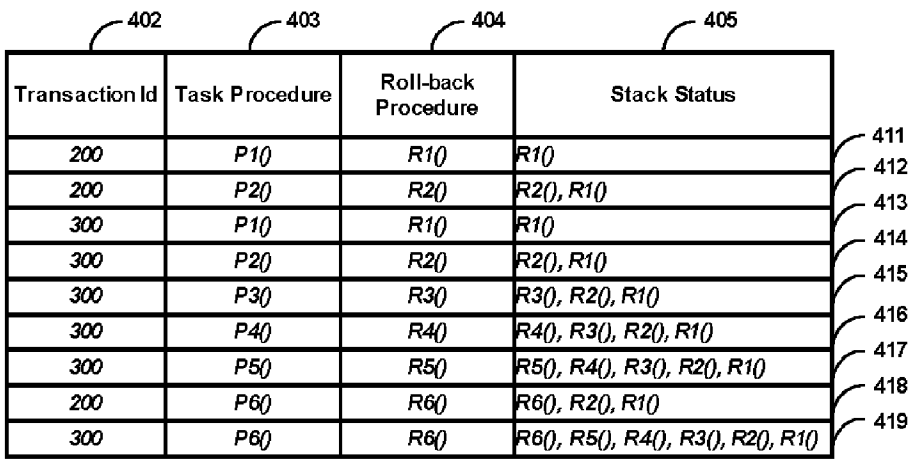
402
403
404
405
Transaction Id
Task Procedure
Roll-back Procedure
Stack Status
200
P1()
R1()
R1()
411
200
P2()
R2()
R2(), R1()
412
300
P1()
R1()
R1()
413
300
P2()
R2()
R2(), R1()
414
300
P3()
R3()
R3(), R2(), R1()
415
300
P4()
R4()
R4(), R3(), R2(), R1()
416
300
P5()
R5()
R5(), R4(), R3(), R2(), R1()
417
200
P6()
R6()
R6(), R2(), R1()
418
300
P6()
R6()
R6(), R5(), R4(), R3(), R2(), R1()
419

FIG. 4

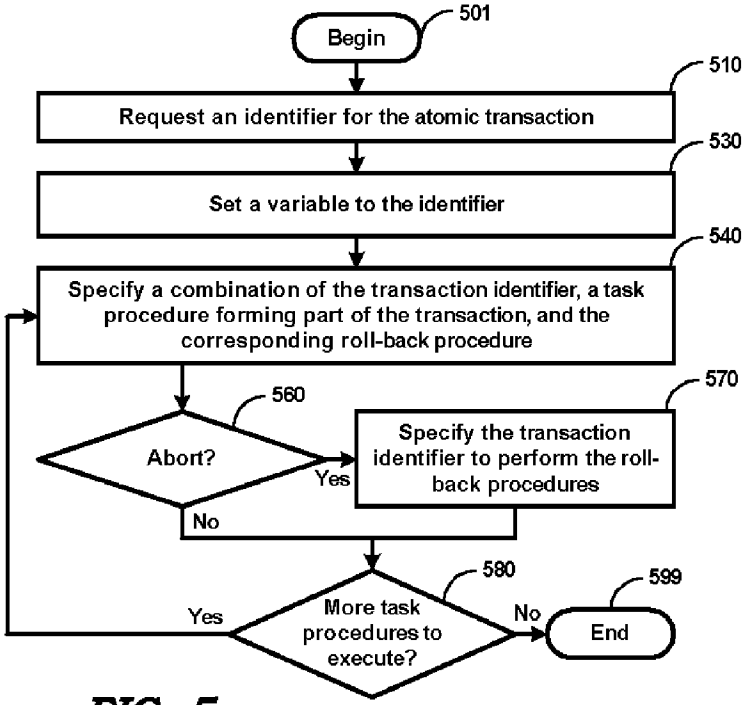
Begin
501
Request an identifier for the atomic transaction
510
Set a variable to the identifier
530
Specify a combination of the transaction identifier, a task procedure forming part of the transaction, and the corresponding roll-back procedure
540
Abort?
560
Yes
No
Specify the transaction identifier to perform the roll-back procedures
570
More task procedures to execute?
580
Yes
No
End
599

FIG. 5

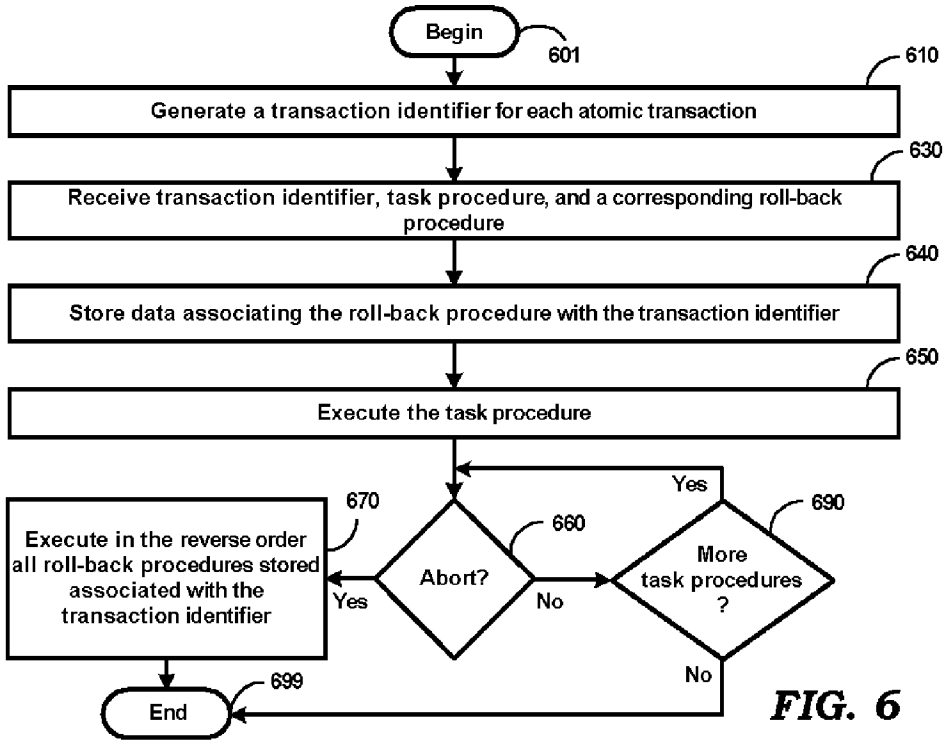
Begin
601
610
Generate a transaction identifier for each atomic transaction
630
Receive transaction identifier, task procedure, and a corresponding roll-back procedure
640
Store data associating the roll-back procedure with the transaction identifier
650
Execute the task procedure
Yes
690
More task procedures ?
No
660
Abort?
Yes
No
670
Execute in the reverse order all roll-back procedures stored associated with the transaction identifier
699
End

FIG. 6

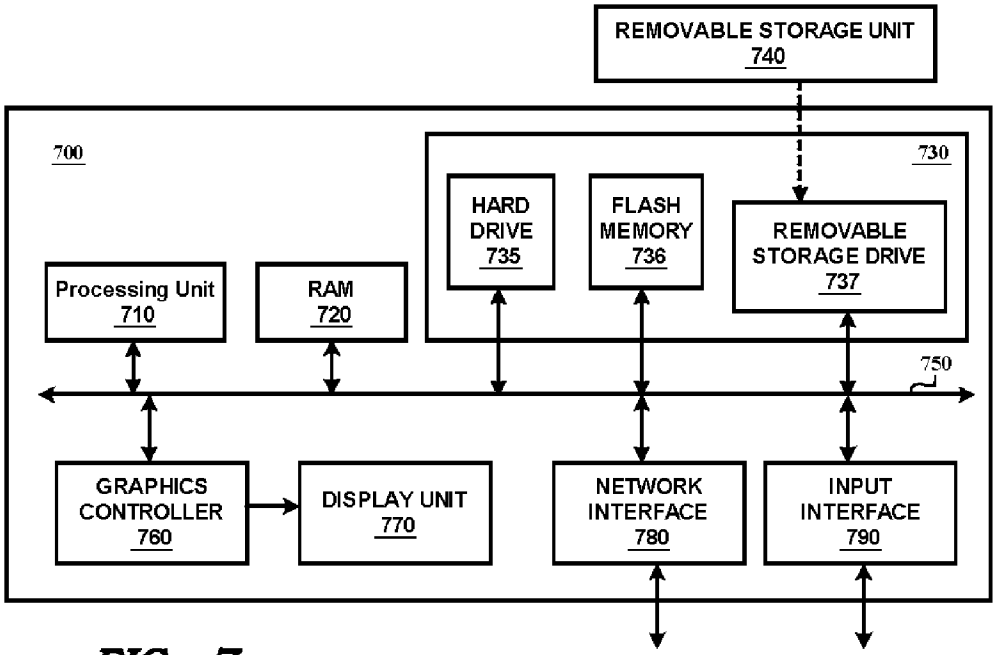
REMOVABLE STORAGE UNIT
740
700
730
HARD
DRIVE
735
FLASH
MEMORY
736
REMOVABLE
STORAGE DRIVE
737
Processing Unit
710
RAM
720
750
GRAPHICS
CONTROLLER
760
DISPLAY UNIT
770
NETWORK
INTERFACE
780
INPUT
INTERFACE
790

FIG. 7